Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall , Year:2025), B.Sc. in CSE (Day)

Lab Report NO: 02

Course Title: Artificial Intelligence lab

Course Code: CSE 316 Section:231 D3

Lab Experiment Name: Implement Graph Coloring Algorithm & Solve N-Queen problem Using Backtracking Algorithm.

Student Details

Name		ID
1.	Promod Chandra Das	231002005

Lab Date : **10.12.25**

Submission Date : **17.12.25**

Course Teacher's Name : **Mr.Mozdaher Abdul Quader**

Lab Report Status

Marks: ………………………………… **Signature:....................**

Comments:.. **Date:.............................**

1.TITLE OF THE LAB REPORT EXPERIMENT

Implement Graph Coloring Algorithm & Solve N-Queen problem Using Backtracking Algorithm.

2. OBJECTIVES

The objective of this experiment is to study and implement the backtracking algorithm for solving constraint satisfaction problems. This experiment aims to apply the backtracking technique to solve the Graph Coloring problem by assigning colors to vertices such that no two adjacent vertices have the same color. It also aims to solve the N-Queen problem by placing N queens on an N×N chessboard in such a way that no two queens attack each other. Through this experiment, the use of recursion, constraint checking, and systematic backtracking is understood.

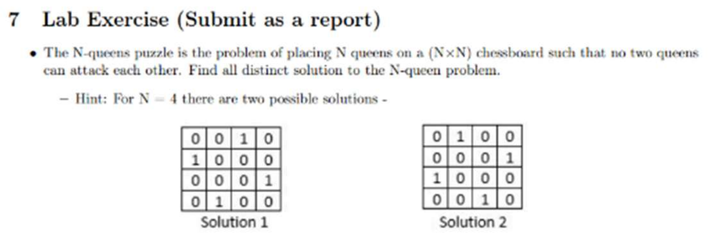
3.PROCEDURE

Initially, the problem is analyzed to identify the constraints that must be satisfied. For the Graph Coloring problem, the graph is represented using an adjacency matrix, and all vertices are initially assigned no color. Colors are then assigned to vertices one by one using the backtracking approach. After assigning a color to a vertex, a constraint check is performed to ensure that no adjacent vertex has the same color. If a conflict occurs, the assigned color is removed and another color is tried. This process continues recursively until all vertices are colored or no valid solution exists.

For the N-Queen problem, an empty N×N chessboard is considered, and queens are placed column by column. For each column, every row is checked to determine whether placing a queen at that position is safe. The safety check ensures that no other queen exists in the same row or along the diagonals. If a safe position is found, the queen is placed and the algorithm proceeds to the next column. If no safe position is available in a column, the algorithm backtracks by removing the previously placed queen and tries an alternative position. This process continues until all queens are successfully placed on the board.

4.IMPLEMENTATION

4.1



7 Lab Exercise (Submit as a report)

- The N-queens puzzle is the problem of placing N queens on a (N×N) chessboard such that no two queens can attack each other. Find all distinct solution to the N-queen problem.
  - Hint: For N = 4 there are two possible solutions -

0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0

Solution 1

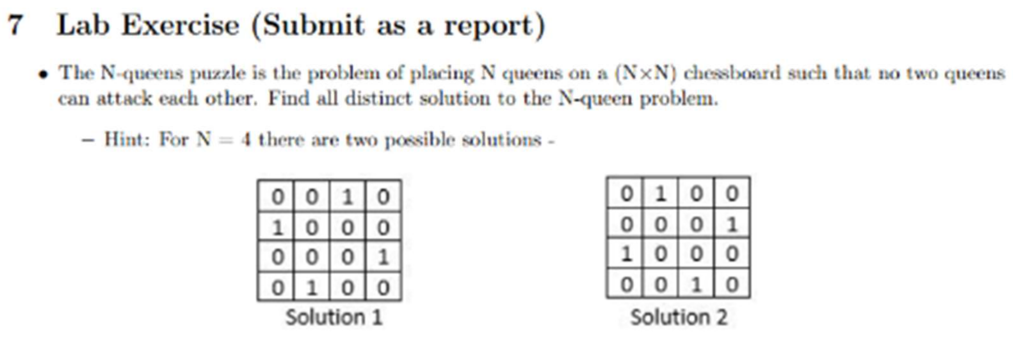
0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

Solution 2

Answer:

```
def is_safe(board, row, col, n):
    for i in range(col):
        if board[row][i] == 1:
            return False

    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1

    i, j = row, col
    while i < n and j >= 0:
        if board[i][j] == 1:
            return False
        i += 1
        j -= 1

    return True


def solve_nqueen_util(board, col, n):
    if col == n:
        return True

    for row in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            if solve_nqueen_util(board, col + 1, n):
                return True
            board[row][col] = 0
    return False


def solve_nqueen(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    if solve_nqueen_util(board, 0, n):
        for row in board:
            print(row)
    else:
        print("No solution exists")



# Example
solve_nqueen(4)
```

4.2



7 Lab Exercise (Submit as a report)

- The N-queens puzzle is the problem of placing N queens on a (N×N) chessboard such that no two queens can attack each other. Find all distinct solution to the N-queen problem.
  - Hint: For N = 4 there are two possible solutions -

0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0

Solution 1

0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

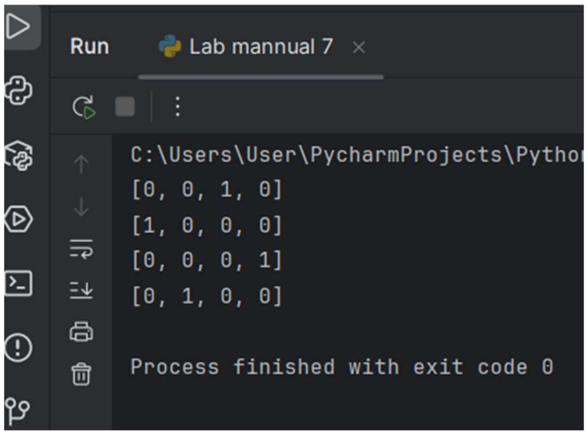
Solution 2

Answer:

```
def is_safe(row, col, pos):
    for r in range(row):
        c = pos[r]
        if c == col:  # same column
            return False
        if abs(c - col) == abs(r - row):  # diagonal clash
            return False
    return True

def build_matrix(pos, n):
    board = [[0] * n for _ in range(n)]
    for r in range(n):
        board[r][pos[r]] = 1
    return board

def solve_all_nqueens(n):
    pos = [-1] * n
    solutions = []

    def backtrack(row):
        if row == n:
            solutions.append(build_matrix(pos, n))
            return
        for col in range(n):
            if is_safe(row, col, pos):
                pos[row] = col
                backtrack(row + 1)
                pos[row] = -1  # backtrack

    backtrack(0)

    # Print solutions in the required format
    for idx, sol in enumerate(solutions, 1):
        print(f"Solution {idx}")
        for r in sol:
            print(" ".join(map(str, r)))
        print()

    print("Total distinct solutions:", len(solutions))

# For the lab hint case:
solve_all_nqueens(4)
```

5.TEST RESULT / OUTPUT

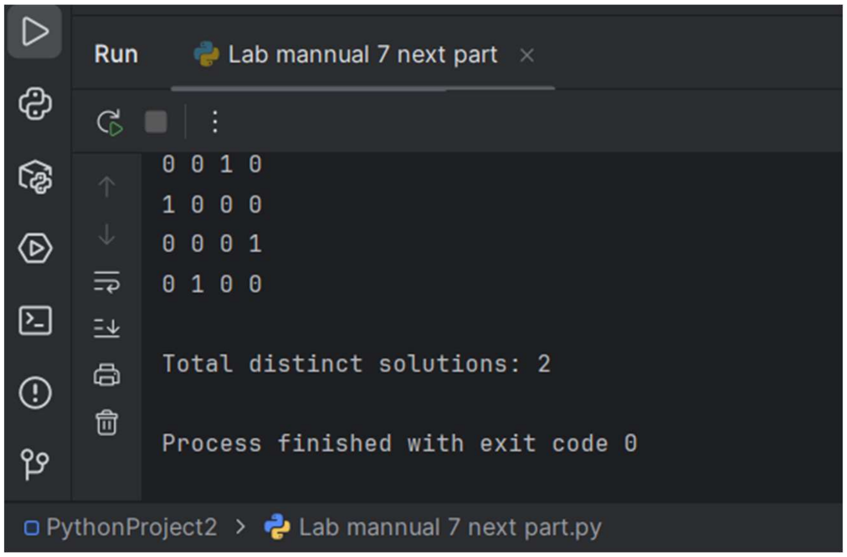
4 .1 : OUTPUT



```
C:\Users\User\PycharmProjects\Pytho
[0, 0, 1, 0]
[1, 0, 0, 0]
[0, 0, 0, 1]
[0, 1, 0, 0]

Process finished with exit code 0
```

Fiqure: Result/ Output 4.1

4.2: OUTPUT



```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

Total distinct solutions: 2

Process finished with exit code 0
```

Fiqure: Result/Output 4.2

6.ANALYSIS AND DISCUSSION

The N-Queen problem is a classic example of a constraint satisfaction problem where a brute-force approach would require checking all possible arrangements of queens on the chessboard, resulting in extremely high computational cost. The backtracking algorithm significantly reduces this complexity by eliminating invalid solutions at an early stage through constraint checking. By placing queens one row at a time and verifying column and diagonal conflicts before proceeding further, the algorithm avoids unnecessary exploration of impossible configurations.
Although backtracking improves efficiency compared to brute force, the time complexity of the N-Queen problem remains exponential in nature, increasing rapidly as the value of N increases. For small values of N, such as N = 4, the algorithm performs efficiently and successfully generates all distinct solutions. However, as N grows larger, the number of recursive calls increases substantially, making the algorithm less practical without further optimization techniques such as heuristics or symmetry reduction. Despite this limitation, backtracking remains a reliable and systematic method for generating all valid solutions while maintaining correctness.

7.SUMMARY

In this experiment, the N-Queen problem was successfully solved using the backtracking algorithm. The algorithm systematically explored all possible placements of queens on an N×N chessboard while ensuring that no two queens attacked each other. By applying constraint checks at each step and using backtracking to discard invalid configurations, all distinct solutions for the given value of N were generated efficiently. This experiment demonstrated the effectiveness of backtracking in solving constraint satisfaction problems and highlighted its suitability for problems where exhaustive search with pruning is required.